GIER SYSTEM LIBRARY

Order No.: 475

Class: 2.0

Type: Report

Author: P. Lindgreen

Ed.: June 1967 (E)

S A M B A

System til Administration af MagnetBaand i Algol

A/S REGNECENTRALEN

A/S REGNECENTRALEN

June 1967
PL/LK

S A M B A

(System til Administration af MagnetBånd i Algol)

This manual describes a tape-oriented input/output system which is fully integrated with the GIER ALGOL 4 compiler. The system allows the user to express solutions to administrative data processing problems involving record structures and file processing entirely in ALGOL and supplies moreover a number of useful features and facilities not available in other existing systems for GIER.

References:

1) The Buffer and the AMPEX TM-7 Tape Unit. GIER System Library No. 290
2) A Manual of GIER ALGOL 4, Regnecentralen 1967

CONTENTS

1. Background

It has often been claimed that GIER cannot be used effectively for administrative data processing (ADP), because of the small core store. This may have been ~~the truth~~ true until the first quarter of 1963, when the buffer store and tape units were made available for the GIER user; but from that time at least four service-centres entirely based on GIER-systems have managed to prove the opposite.

However, the development of software systems to GIER has unfortunately progressed in such a direction that the programming of ADP for GIER is still unnecessarily laborious.

The successful GIER ALGOL III system which was developed during 1963 and released in February 1964 completely ignored the existence of the buffer itself and the tape-units connected hereto, while on the other hand it contained a nearly complete set of useful standard procedures for the handling of input/output on paper tape.

In January 1967, a preliminary version of the GIER ALGOL IV system was released. Compared to previous versions, this system seems to be much more powerful and effective, primarily because of its use of the buffer for storing arrays, but also because many operations may now be done in a faster or more useful way.

However, tape units as an external medium are still looked upon as 'The Ugly Duckling'. Only two extremely primitive and for administrative jobs probably practical useless standard procedures are delivered with the system. They just activate the two machine instructions IL and US. This must be compared with the fact that so highly advanced standard procedures as e.g. read general, put, get, write etc. are considered reasonable in the system. why?!

The consequence of the above mentioned policy has been that nearly all ADP-programs for GIER have been programmed in machine code based on more or less restricted input/output systems. This has given the installations a great deal of problems with respect to the documentation and maintenance of the programs and may generally be considered old-fashioned compared to the present state of art concerning use of programming languages.

In June 1965, the present writer proposed that the problems should be solved by a set of standard procedures for handling data files which were to be implemented under GIER ALGOL III. The proposal was based on experience gained through three years' work with compiler construction and from two major ADP-projects involving the use of a very convenient I/O-system developed by Regnecentralen for the CDC 1604.

The proposal was based on the following philosophy:

1) The system should contain sufficient facilities to avoid use of machine code.

2) The user should have possibility to declare a number of records through a procedure call, which should connect an arbitrary number of individual ALGOL-variables to one single identifier. Moreover he should have possibility to get access to the records of a file through simple procedure calls using the identifier mentioned as only parameter describing the record.

3) Any tape error and the recognition of EOF or EOT should result in activating a declared ALGOL-procedure, allowing the user to take any action on the actual situation including means for communicating to the system, which action it should overtake upon return from the user procedure.

Besides these principles, the system should contain convenient tape protection facilities and an effective double buffer administration where the buffer areas should be anonymous from the ALGOL program.

The proposal was rejected - primarily caused by opposition from the so-called GIER-programming committee, which was at that time considered the only authority in this field. Then nothing happened until August 1966 when it became obvious that the official GIER ALGOL IV system would not support the users of tape units.

At that time the above mentioned proposal was revised and made compatible with GIER ALGOL IV using the new features available, but without changing the basic philosophy. A working group consisting of members

from RC's service centres was formed, and after some discussions - which took place in the period August-October 1966 - it was possible to design the system which got the name SAMBA to such an extent that the programming and final design could begin in November 1966.

Primarily the the disign, programming and debugging of the system have been done by Kim Andersen and Inger Møller from Århus and by Bjørn Ørding-Thomsen, Flemming Sylvest Pedersen and the present writer from Copenhagen, but also other persons - expecially members of the compiler group - have supported the project in different ways primarily by proposals or by criticism.

In this connection, it has been of great importance for the result obtained that quite different meanings about details of the system have been expressed by persons either directly or indirectly involved in the project. In many cases, this has postponed the finishing of the system but it may have helped us to look upon the problems in a more general way and thereby to evaluate better solutions.

I want to express my warmest thanks to all persons who have contributed to the accomplishment of the project one way or another.

A/S REGNECENTRALEN
June 1967

Paul Lindgreen

2. Survey of the system

Scope : SAMBA is a set of standard procedures to GIER ALGOL 4 allowing the user to generate and get access to data files on magnetic tape ~~in a very convenient and useful way~~.

Flexibility : SAMBA provides the user with all the functions and operations which are common to and a necessary part of most programs dealing with magnetic tape files, but the system is integrated with GIER ALGOL 4 in a way which makes it ~~very~~ easy for the user to modify, expand or combine many of the functions.

Tape protection : No tape operation on a given tape is possible within SAMBA until the first block on the tape (normally a special tape label) has been checked. The check is performed according to information specified as parameters to a tape-test-procedure and to rules which are introduced into the system when the compiler tape is generated.

Data files : In SAMBA the user is allowed to operate with up to 1023 data files (separated by filemarks) on the same tape. Moreover SAMBA provides the user with means to program a simple administration in ALGOL for data files which may expand to more than one tapereel.

File labels : A data file on a SAMBA-generated tape is identified by the contents of the first block, the so-called file label. The file label contains the number of the file and information to the system about maximum block size, mode etc. Besides this fixed part, the user may provide or get access to an optional number of words in which he may store any information concerning the actual file.

Record-declarations : In SAMBA the user may declare a number of records through special calls. A record may consist of one or more ALGOL-variables - simple as well as subscripted - in any order and of any type. The record is identified by the name of an ALGOL variable which contains a reference to an internal table describing the structure of the record.

Double buffer administration : The records declared, which may be common to more files, are read or written on specified tape units via a double buffer administration using anonymous buffer areas - one area for each open file. A file is opened through a call of a special procedure which looks up the current file on the tape, reserves the necessary buffer area and processes the information to or from the file label.

Checksum : When an output file is opened, the user may specify whether he wants a checksum of each record generated and whether he wants it to be stored in an anonymous extra word of the record when it is output. Information about the presence of this extra word will be stored in the file label.

When such a file is opened as inputfile, the user may specify whether he wants the generated sum checked on input of each record.

Independent of the presence of an extra checksum-word, the user may get the computed checksum after each record-operation as the contents of a standard variable named sambasum. In this way the user is able to generate and check a hash-total for each file.

Variable record-length : When the extra record-word mentioned above is present, the actual length of the record is stored there, independent of a possible checksum. This means that the user may operate with records with variable length, and SAMBA contains the necessary procedures herefor.

Error-check-facilities : In SAMBA every tape transport is checked in the most thorough manner in order to detect any possible errors. The introduction of the so-called block-label - two extra words connected to each data block in a file - makes this detection easy. The block-label contains the number of the current block inside the file and the length of the block.

When a tape error is detected, SAMBA automatically tries to recover it by a simple reread or rewrite operation. If the error is not removed hereby, a user declared interrupt procedure - specified to the system when the current file has been opened - is called with the necessary information to allow the user to take any legal ALGOL-action in this situation.

Upon return, the user may communicate to the system whether he wants the error to be ignored, whether a new attempt to reread/rewrite should be activated, or whether he wants the erroneous data skipped. If no interrupt procedure is specified, the error will normally be ignored.

Besides the treatment of tape errors described above, the system is almost foolproof with respect to incorrect use of the procedures. When such errors are detected, the normal GIER ALGOL 4 error procedure will be activated.

Reaction on end of file : When a file mark is recognized on an input file, the user declared error procedure mentioned above is activated directly without any recovery attempt, but with the information that the current 'error' is no error but EOF.

Optional tape formats : As it is seen from the description given so far, SAMBA ~~is a very~~ tries to be a safe and user-friendly system with many facilities which are not normally available.

The price for this is so strong demands to the tape format that tapes generated with the standard facilities in SAMBA are ~~completely~~ incompatible with all other systems.

This is of course no restriction as long as the programmes do not require any communication with other computers, but under certain circumstances this may be the case.

In order to make this communication possible without sabotaging SAMBA's own tape security, a set of simple procedures are provided, allowing the user to generate, read and check tapes with any format, but on the other hand requiring that he programs the necessary administration himself.

Hardware maintenance facilities : The detection of a tape error - which cannot be removed after the build-in automatic recover action - causes SAMBA to count up by one an internal counter corresponding to the current unit. All counters are initialized to zero before the run of a given program, but it is up to the surrounding system - which uses the GIER ALGOL 4 system as a subroutine - to take the proper action on the information available in these counters.

3. The SAMBA-tape format

The field of datamatics which is covered by the SAMBA-system may be described as: Handling of information organized in data files on magnetic tapes. The process on the information will in most cases be rather simple, but will involve a great amount of uniform structured data (records).

With such problems it is of great importance that the tape-handling system is reliable in order to avoid that unrecognized errors cause the data to be wrong. This has been obtained in SAMBA by means of a very effective error handling system described in section 7 and through the introduction of the special SAMBA tape format.

A SAMBA tape is formed by a tape label (cf. 3.1.) followed by a file mark and up to 1022 data files (cf. 3.2.).

3.1. Tape labels and tape protection

A basic assumption in the design of SAMBA has been the existence of an external administrative system on each installation which defines and maintains the rules for use of magnetic tapes and which is able to secure that a special tape label is always written on a tape before it is made available for the user of SAMBA.

The purpose of the tape label is to make it possible for the programmer through the use of SAMBA:

a) to make sure that the correct tapes are mounted when the program is running,
b) to give a reasonable protection of the information on tapes involved in a project.

A tape label in SAMBA consists of three groups of information:

Field 0: Four characters which classify the tape.
Field 1: Four decimal characters which form the number of the tape and identify it in the external administrative system.
Field 2: A pattern consisting of 40 bits which form a further identification of the tape and which are normally known only by the specific user.

Tape-classes

Three tape-classes are distinguised by SAMBA:

If the classification field (field 0) contains the characters GIER it defines the tape to be one with the standard SAMBA-format (cf. appendix A).

If the classification field contains the characters RCAR, the tape is defined as a working tape with no definite format. In this case the contents of field 2 will be irrelevant since a working tape will normally be common to more users.

In all other cases the tape is considered to be used for communication to systems with other tape formats, and activation of the normal SAMBA procedures will have no meaning. In these cases the more primitive procedures for block transports should be used (see description of procedures blockin, blockout and status).

Tape identification

Before any operation in SAMBA on a given tape is allowed to take place, a special tape-label-checking procedure must be called (see description of procedure tapetest). In order to make tape processing legal and thereby possible, information supplied as parameters to this procedure should match the tape-identifying fields (1 and 2) in the tape label according to the information in the classification field and to rules which are defined in a table available for the procedure.

Definition of the rules for checking

The table entry to be used depends primarily on the class of the tape and moreover on the results of the identification process. The contents of the table (and thereby the rules) are defined in the standard version of the SAMBA-system as described in appendix D. However, since some installations may have a policy for what should be allowed in certain cases which differ from that expressed in the standard version, a formal procedure for ordering a SAMBA-system with specified rules has been established. This procedure is described in appendix E.

3.2. Data files and file labels

A data file (or just file) is defined as the information between any two file marks on the tape. It is formed by a file-label-block and a number of data-blocks.

The file label consists of a fixed system-part and an optional user-part. The fixed part contains the number of the current file (first file after the tape label has number one), information about maximum block size inside the file, whether the records were generated with an extra checksumword and whether they were written with 4 or 7 characters pr. word. This information is used by the system when an input-file is opened to reserve the necessary buffer area and to secure that the correct read-block-instructions are used etc. (see description of the procedures openin and openout).

The optional part of the file label - which may consist of up to 8 words - is available for the user for any purpose. It may be used for further identification of the file, for storing special protection information, e.g. retention cycle, and for accumulated data, e.g. a hash-total, for the previous file on the same tape.

The last mentioned use is possible, because the last data file on a tape will always be followed by a dummy file only containing a file label block. The dummy file label is generated in the procedure close, which must be called as a termination of the process on the current file.

Tapes which are used by the system for the first time should have the dummy file label generated together with the tape-label in a previous special process. This dummy label is moreover used by the system to secure that references are only made to files really existing on the tape (cf. description of procedure getfile, 6.3.2.).

3.3. Data blocks, block labels and records

The data blocks of a file consist - like any other information on the tape - of a number of characters, but are considered by the system a number of GIER-words either 4 or 7 characters each, as specified to the system when an output-file is opened. The two first words of a block - the so-called block-label - are generated and used

by the system as an identification of the block. They contain the number of the block inside the current file and the length of the block (including the two words of the block-label).

The remaining words of the block form the records, i.e. the logical units of information which the user must specify to the system in the input/output statements through which he transfers the information between the files and the variables of the program or vice versa (see the description of procedures inrec, outrec, invar and outvar).

The system will increase the number of words in a record by one, if the user specifies this when an output-file is opened. In the extra word, the length and - if required - the checksum of the current record are stored. When such a file is opened as input-file later on, the user may specify whether he wants the generated sum checked during the input-operation.

4. Records and record-declarations

4.1. The record concept

A data file may be defined as a sequence of logical information units - records - which may differ both with respect to format and contents. In practice, however, the number of different record-formats considered in a given process is very limited, and very often a certain record-format may happen to be common to more files.

When data files are processed by a program running on a computer with word-structure, the elements of a record will normally be equivalent to words - independent of whether the information stored in the record are numbers or text. When the program is an ALGOL-program, this means that the elements are ALGOL-variables.

It would now be very convenient if the elements of a record were allowed to be of any kind, i.e. simple as well as subscripted variables, and to be of any type, since all kinds of actions on the elements, i.e. all kinds of ALGOL-statements using the variables may be of interest in a given process. On the other hand when a record is transferred from the program to a file or vice-versa, it is convenient to look upon it as a simple entity, both with respect to the logic of the program and to the number of individual statements or parameters in procedure calls.

However, ALGOL 60 contains no means to specify links between individual variables or to declare more advanced structures than arrays. Arrays of course could be used, but only resulting in unnecessary laborious programming and too slow processing.

4.2. Record declarations

In SAMBA the facility of declaring a record is provided by means of a standard procedure, which connects an unlimited number of simple and subscripted variables to one single variable, which identifies the record in the input/output statements (see description of procedure record).

It is possible to have one or more elements of a record to be part of another record or to use the same record both for input and output on the same time. These features are very useful, e.g. in processes concerning the updating of files.

5. Internal tables and use of the buffer store

The number of tape-units available for the system is normally 4, but this number may be changed when the translator-tape including the SAMBA-system is generated (cf. appendix E).

The various parameters describing the current use and status of the tape units are stored in a corresponding number of tables - known as the file tables - each occupying 8 words in the buffer store from address 1 onwards.

When a file is opened by call of the procedures openin or openout, a data-buffer-area with length = 2 × maxblocklength + 3 [x] is reserved from the current first free word in the lower part of the buffer. (This current first free will at any time also be used as lower-buf-lim in the array declaration routine of ALGOL).

The reservation of a buffer-store-area will further take place whenever a record is declared by a call of the procedure record. In this area, the necessary information will be stored to make the transfers between the data-buffer and the elements of the record as fast as possible (see description of procedure record). The length of the area is strongly dependent on the structure of the record.

When a file is closed by call of the procedure close, the corresponding data-buffer-area is released. This is also the case with the table defining a given record, when the record is redefined (see description of the procedure record). In both cases, the areas following the released area are moved so that the occupied area used by SAMBA at any time is as compact as possible. The corresponding tables or record identifiers and the current first free pointer are then updated according to the rearrangement of the areas.

In order to help the user utilize the available buffer store area in the best way, any procedure changing it will have as its value the number of remaining free words in the buffer.

x) The three extra words are used as an extension of the file table. Word 1 contains a link to the file table and the length of the reserved area. This information is used by the procedure close. Word 2 contains a specification of the so-called interrupt procedure as described in section 7. Word 3 contains the current tape number (from field 1 in the tape label).

6. Description of the procedures

6.1. Survey

The procedures which form the SAMBA-system may be divided into two groups:

a: Proper SAMBA procedures.

b: Procedures for basic tape operations.

Proper SAMBA procedures

tapetest (6.3.1) — Used to check the label of a tape mounted on a given unit (cf. section 2). Must be called before the activation of any other procedure which operates on the tape.

getfile (6.3.2) — Looks up a certain file on a specified unit and delivers the contents of the file label.

openin (6.3.3) — The same as for getfile, but will moreover reserve a sufficient buffer-area and initialize the file-table for the unit so that the record input procedures inrec and invar may be used.

openout (6.3.4) — The same as for getfile, but will moreover reserve a specified buffer-area and initialize the file-table for the unit so that the record output procedures outrec and outvar may be used.

record (6.3.5) — Used as a means to declare a record area and identify it by a simple variable, which is used in connection with the procedures inrec and outrec.

inrec (6.3.6) — Transfers a record from an open input file to a declared area.

outrec (6.3.7) — Transfers a record from a declared area to an open output file.

invar (6.3.8) — Transfers a record from an open input file to an array and delivers the actual length of the record.

outvar (6.3.9) — Transfers a specified number of words from an array as a record to an open output file.

setproc (6.3.10) — Specifies an ALGOL-procedure to be activated by the system in case of tape errors or EOF/EOT.

research (6.3.11) — Looks up a certain record inside an open file.

close (6.3.12) — Releases the buffer area reserved by openin or openout and transfers information to or from the following file label.

torec (6.3.13) — Moves information from a specified part of an array to a declared record.

fromrec (6.3.14) — Moves the information in a declared record to a specified part of an array.

Procedures for basic tape operations

blockin (6.3.15) — Transfers a data block from a tape unit to an array.

blockout (6.3.16) — Transfers a data block from an array to a tape unit.

status (6.3.17) — Provides information about the result of the last activated call of blockin or blockout.

rewind (6.3.18) — Drives a specified tape to load point.

unload (6.3.19) — The same as for rewind, but will set the tape unit to local control and wind off the tape [x].

drive (6.3.20) — Supplies the remaining basic tape operations such as skip file, backspace etc.

[x] This may depend on the tape unit.

6.2. Legality

The procedures mentioned above are standard procedures to the SAMBA version of GIER ALGOL 4, which means that they may be called from all over an ALGOL-program without any declaration. However, the SAMBA-system itself gives some restrictions on their use.

The legality of a given procedure on a given unit is mainly controlled by the so-called tapestatus and filestatus, which are in fact represented as a number of booleans stored in the filetables described in section 5.

Tapestatus expresses the result of a previous call of the procedure tape-test with respect to which procedures may act on a given tape.

Four independent booleans are used to carry out this control:

Allowed for openin
Allowed for openout
Allowed for blockin
Allowed for blockout.

Filestatus expresses a state for each unit which controls the legality of a given procedure after it has been checked against tapestatus. The possible three values of filestatus are:

0: Closed
1: Open for output (i.e. allowed for outrec and outvar)
2: Open for input (i.e. allowed for inrec and invar).

Filestatus is changed only by the procedures openin, openout and close [x].

Each description - as it appears in the following - will contain a specification of the values of tapestatus and filestatus which are necessary to make the use of the procedure legal.

If nevertheless a procedure which is illegal is used in a given situation, the run of the program will be terminated with a standard GIER ALGOL 4 error message, because it indicates a formal error in the program (cf. section 7 and appendix B).

[x] In fact there is one more value saying: Neither open nor closed. This value is set internal by the double buffer administration to indicate that a file mark has been read or an end-of-tape mark has been sensed, but a call of close (which is necessary in such a case) has not yet been activated (cf. section 7.3.).

6.3.1

integer procedure tapetest (unit, ident1, ident2, tapeuse, rewind);

Legality Tapestatus: Irrelevant
Filestatus: Closed

Function The tape on the unit specified as first parameter is rewound to load point and the first block, which is assumed to be a tape label, is read.
The integer value specified as second parameter is converted to a four digit decimal number with zero represented by the character value 12 (octal) which is compared with the tape number field in the tape label (cf. section 3.1.).
The value of the boolean third parameter interpreted as a bit pattern is compared to the third field of the tape label.
The result of these two comparisons is used to evaluate the value of the procedure and the so-called tapestatus which defines the procedures which are allowed to be used as long as the current tape is mounted. The rules for this evaluation are described below.
The integer value of the fourth parameter is compared with the integer value composed by the two bits of the statusword which specifies write-protection-ring present on the tape and high/low-density on the tape unit.
If - and only if - the two values are not matching, the following message to the operator is given on the typewriter:

unit <unit> rd <tapeuse>

where <unit> and <tapeuse> are the values of the first and fourth parameters respectively, and where rd is an abbreviation of ring and density.
The meaning of the relevant values for tapeuse is:

0	remove ring	set low density
1	remove ring	set high density
2	ring wanted	set low density
3	ring wanted	set high density.

Having printed the message, GIER will stop and wait for a reaction from the operator. When the START-button on the main panel is pushed, a re-entry in tapetest will take place

and all actions and tests mentioned above will be repeated. If the tapeuse-test shows ok, the last parameter is considered. If its value is zero or negative, the procedure returns with no further action. If the parameter is greater than zero, the current tape will be rewound to load point. (The latter action will normally only have meaning, if the tape has a format which differs from the SAMBA-format.)

Value

The value of tapetest may be described by the following statements which express the basic logic of tapetest:

```
rewind (unit);
read block uneven parity and indicate tapeerror;
field 0:= first 4 characters of block;
field 1:= next 4 characters of block;
field 2:= next 40 bits of block;
class:= if field 0 = ⟨ GIER ⟩ then 1
else if field 0 = ⟨ RCAR ⟩ then 2 else 3;
ok1:= if field 1 = ident1 then 0 else 1;
ok2:= if field 2 = ident2 then 0 else 2;
value:= case class of ok1 + ok2, ok1, 4 + ok1 + ok2;
tapetest:= if tape error then - value else value;
```

Expressed in another way:
If the value is zero or four, the tape has been recognized according to the specified identification.
If the value is greater than or equal to 4, the tape has no standard SAMBA label.
If it differs from 0 or 4, either ident1 or ident2 or both were not matching their respective fields in the label.
If the value is negative, it indicates a tape error which may have caused a possible non-coincidence.
If the tape is recognized as a working tape (field 1 = RCAR), the comparison with field 2 is suppressed, and the value of ident2 will therefore be irrelevant to tapetest.

Tapestatus

As mentioned in section 3.1., the rules,for which procedures are allowed to be used in connection with a given result of tapetest,are stored in an internal table. The entry in this table is dependent on the class of the tape, on the result of the identification process and on the last parameter. The rules defined for the standard version of SAMBA are described in appendix D.

6.3.2

integer procedure getfile (unit, fileno, labelarray);

Legality — Tapestatus: Allowed for openin or openout
Filestatus: Closed

Function — The tape on the unit specified by the first parameter is scanned until the label of the file with number specified as second parameter has been read x).
The contents of the user accessible part of the file label (see section 3) are transferred to the first (max. 8) words of the array specified as third parameter.

Value — The value of getfile indicates to the user the result of the call in the following way:
getfile $\geq$ 0 => The file wanted has been found, and the value is the number of free words in the buffer store.
getfile < 0 => The file wanted has not been found, and the position on the tape is undefined. (This may be caused by irrecoverable tape errors when reading the file labels, by wrong tape format, or if another tape has been mounted on the unit without a call of tapetest in an attempt to compromise the tape security.)

Filestatus — Unchanged.

x) The search for the file is performed in the following way:

```
count:= 0; curfile:= value of current file from internal filetable;
A: if fileno ≤ curfile then
   begin skipfile backward (unit); curfile:= curfile -1; goto A end;
B: skipfile forward (unit); curfile:= curfile +1; read file label;
if curfile ≠ fileno in label then
begin count:= count +1;
if count > 3 then exit with value = -curfile
   else begin rewind (unit); curfile:= 0; end
end;
if fileno > curfile then goto B;
```

6.3.3

integer procedure openin (unit, fileno, labelarray, check);

Legality — Tapestatus: Allowed for openin
Filestatus: Closed

Function — The tape on the unit specified as first parameter is scanned until the label of the file with number specified as second parameter has been read. (This is done by internal use of the procedure getfile.)
The information in the first part of the file label about maximum blocklength of the file (cf. section 3.2.) is used for reservation of the necessary double buffer area in the buffer store.
The file table corresponding to the current unit is initialized according hereto and to the information about mode also contained in the first part of the file label. The initialization of the table will further be controlled by the value of the last parameter, which specifies whether a checksum must be computed during each input operation.
The possible values are:

check = 0 => no checksum
check = 1 => checksum computed.

The contents of the second part of the file label are transferred to the first (max. 8) words of the array specified as third parameter.
Finally, before return the reading of the first data block will be activated.

Value — The value of openin indicates to the user the result of the call in the following way:
openin $\geq$ 0 => The file wanted has been opened for input. The value is the number of free words in the buffer store.
openin < 0 => The file wanted has not been found, and the position on the tape is undefined (cf. value of getfile).

Filestatus — openin $\geq$ 0 : Open for input
openin < 0 : Closed.

6.3.4

integer procedure openout (unit, fileno, labelarray, area, mode);

Legality — Tapestatus: Allowed for openout
Filestatus: Closed

Function — The tape on the unit specified as first parameter is scanned and positioned just before the label of the file with number as specified as the second parameter. (This is done by internal use of the procedure getfile.)
A double buffer area is reserved in the free part of the buffer store, and the filetable corresponding to the current unit is initialized according hereto. The reservation is based upon the value of the fourth parameter, which specifies the number of words which may be used by the system as data area for the file. This value may never be less than 11 (cf. section 5) x).
The initialization of the table will further be controlled by the value of the last parameter, which specifies whether the records will be written on the tape with 4 or 7 characters pr. word, whether a checksum will be computed during each output operation, and whether an additional checkword will be generated on each record to contain the current record length and a checksum if specified.
The possible values of mode and their meanings are:

mode	char/word	checkword	checksum
0	7	no	no
1	7	no	yes
2	7	yes	no
3	7	yes	yes
4	4	no	no
5	4	no	yes
6	4	yes	no
7	4	yes	yes.

The filenumber and information derived from the parameters area and mode together with the first (max. 8) words of the array specified as third parameter are now written on the current tape as a file-label-block (cf. section 3.2.).

x) Maximum blocklength will be computed to: (area - 3) : 2.

Value

The value of openout indicates to the user the result of the call in the following way:

openout $\geq$ 0 => The file wanted has been opened for output. The value is the number of free words in the buffer store.

openout < 0 => The file wanted could not be found (cf. value of getfile), or it was impossible to write a new file label. The position on the tape is undefined.

Filestatus

Openout $\geq$ 0 : Open for output

Openout < 0 : Closed.

6.3.5

integer procedure record (name, element1, element2, ..., elementn);

Legality Tapestatus: Irrelevant
Filestatus: Irrelevant

Function The procedure acts as a pseudo declaration of a structure of ALGOL-variables known as a record (cf. section 4).
The record will be identified by the boolean variable specified as first parameter. This variable must be reserved for this purpose only, since the procedure will store in it a reference to an internal table describing the structure of the record.
The elements of the record are described in the remaining parameters to the procedure. These elements may be either simple variables, subscripted variables or array identifiers and may be of any type. There are no restrictions to the number of elements in a call as well.
The procedure will build up an internal table in the buffer store (cf. section 5) in which descriptions of the elements of the record are stored. These descriptions have a format which is convenient for the transfers of information to and from the elements (see description of procedures inrec, outrec, torec and fromrec).
The procedure will recognize it if two or more consecutive elements have consecutive addresses in the store and in this case generate a common description of this group of elements, which again means that the transfers will be faster.
One word in the table will be reserved for each description of a simple variable or group of consecutive simple variables, and two words will be necessary for each four consecutive array words x).

Redeclaration If the procedure is called with a variable as first parameter, which has already been used as record-identifier in a previous call, the already existing record description table will be released, and a new one corresponding to the new call will be built up.

x) The user is advised to consider the possibilities for changing this value as described in appendix E.

If - in this case - the procedure is called with the first parameter as the only one, the effect will be a release of the table only.

Value — The value of the procedure expresses the number of free words in the buffer store after the return.

Warning — Since the declaration of a record through a call of this procedure is no real declaration, the translator will have no possibility to check whether the variable specified as first parameter is used as reference to the record in a place of the ALGOL-program which is outside the scope of one or more of the elements.
The user is therefore advised always to declare the variables which form the elements of a record, either global to or in the same block as that in which the identifier of the record is declared.

6.3.6

boolean procedure inrec (unit, name);

Legality — Tapestatus: Allowed for openin
Filestatus: Open for input

Function — The procedure will transfer information from the file opened on the unit specified as first parameter to the elements of a record identified by the boolean variable specified as second parameter. If specified in the open call (through the check-parameter), an arithmetical sum of all the elements of the record (interpreted as integers independent of type) is generated during the transport.
This sum will be stored in the standard variable sambasum and will moreover be checked against a corresponding sum, if such a sum was generated, and stored in an extra word of the record itself, when it was output (cf. description of the procedures outrec and outvar). The extra word - if present - will not be available for the user, but the length contained in it will be checked against the length of the record specified to the procedure.
The transfer from the file will take place via a double buffer administration. This means that the records are taken from two alternating block-buffers, into which the data blocks of the file are read in turn. The status of this unpacking as well as the current blocknumber etc. are stored in the internal file table which was initialized in the open call.
The first block transport will be activated in the open call, and a transport will always be checked before the records of the block are used. Tape errors are treated as described in section 7.

Value — The value of the procedure will normally describe the location of the current record inside the file expressed by the current blocknumber and the relative address of the first element of the record inside the block.

This information is packed in the word representing the value so that the blocknumber occupies the leftmost 20 bits and the blockrelative the rightmost 20 bits.
The value may be stored in any boolean variable and is primarily thought to be used, if the current record must be looked up again later on in the process (see description of procedure research).
The value may, however, be equal to 40 0, i.e. all bits are zero, if no special error procedure has been specified in cases where a file mark is sensed (cf. section 7).
In this situation, no record has been transferred (no one is available), and a possible following attempt to call inrec again on the same unit, before the current file has been closed and a new one opened, will cause the run to be terminated with a normal error message (cf. appendix B).

Warning — See warning in description of procedure record (6.3.5).

6.3.7

boolean procedure outrec (unit, name);

Legality — Tapestatus: Allowed for openout
Filestatus: Open for output

Function — The procedure will transfer information from the elements of a record identified by the boolean variable specified as second parameter to the file opened on the unit specified as first parameter.
If specified so in the open call (through the mode-parameter), information about the length of the current record will be assigned to the record as an extra so-called checkword preceding the words representing the record elements (cf. section 3.3.).
If specified so in the open call, an arithmetical sum of all the elements of the record (interpreted as integers independent of type) is generated during the transport.
This sum is stored in the standard variable sambasum and will moreover be packed in a compressed form together with the record length in the above mentioned extra checkword, if it is present.

The transfer to the file will take place via a double buffer administration. This means that the records are packed in two alternating block-buffers which will be written on the tape when they are filled up. The status of this packing as well as information about the current blocknumber etc. are stored in the internal file table, which was initialized in the open call.
Before a block transport is activated, the previous transport will be tested. Tape errors are treated as described in section 7.

Value — The value of the procefure will normally describe the location of the current record inside the file expressed by the current blocknumber and the relative address of the first

element of the record inside the block. This information is packed in the word representing the value so that the blocknumber occupies the leftmost 20 bits and the blockrelative the rightmost 20 bits.
The value may be stored in any boolean variable and is primarily thought to be used if the current record must be looked up again later on in the process (see description of procedure research).
The value may, however, be equal to 40 0, i.e. all bits are zero, if the EOT-mark was sensed during a writeblock operation and no special error-procedure was specified (cf. section 7).
If the procedure returns with the value 40 0, no record has been transferred, and a possible following attempt to output a record on the current file will cause that the run will be terminated with a normal error-message (cf. appendix B).

Warning — See warning in description of procedure record (6.3.5).

6.3.8

boolean procedure invar (unit, dataarray, length);

Legality — Tapestatus: Allowed for openin
Filestatus: Open for input

Function — The procedure will transfer a record from the file opened on the unit specified as first parameter to the elements of the array specified as second parameter.
The number of words to be transferred will either be derived from the extra checksum/length-word generated during output (mode = 2, 3, 6, 7 in procedure openout), or it must be specified in the integer third parameter if no extra word is available in the records (mode = 0, 1, 4, 5) x).
If the length is greater than the length of the specified array, only as many words are transferred as may fit into the array.
Upon return, the third parameter will in any case contain the real length of the record x), and a following call of invar (or inrec) will give access to the next record of the file.
The procedure will use the same double buffer administration as the procedure inrec and may - if convenient - be used together with this procedure.
The rules for computing the checksum are exactly as described for inrec.
Tape errors and end-of-file are treated as described in section 7.

Value — The value of the procedure is exactly as described for procedure inrec.

x) Length is here defined as the number of words in the record except the extra length- and checksumword.

6.3.9

boolean procedure outvar (unit, dataarray, length);

Legality — Tapestatus: Allowed for openout
Filestatus: Open for output

Function — The procedure will transfer a number of words as specified in the third parameter from the array specified as second parameter as a record to an open output file on the unit specified as first parameter.
If specified so in the open call (through the mode parameter) information about the length of the record is assigned to the record in an extra word.
The rules for computing and storing a checksum are exactly as described for procedure outrec.
The procedure will use the same double buffer administration as outrec and may be used together with this procedure when a file is generated.
Tape errors and end-of-tape are treated as described in section 7.

Value — The value of the procedure is exactly as described for procedure outrec.

6.3.10

procedure setproc (unit, interruptprocedure);

Legality — Tapestatus: Allowed for openin or openout
Filestatus: Open for input or open for output

Function — The procedure will supply the system with a reference to the procedure specified as second parameter. This procedure will be used as interrupt procedure (as described in section 7) for the file opened on the unit specified as first parameter.
If setproc is called more than once during the period a given file is open, the earlier specified interrupt procedure will be substituted by the new one.
The demands to the declaration of the interrupt procedure are described in section 7.2.3.

Warning — Cf. the warning in section 7.2.3.

6.3.11

integer procedure research (unit, filepoint);

Legality — Tapestatus: Allowed for openin or allowed for openout
Filestatus: Open for input or open for output

Function — The procedure will look up a certain point (record) inside the file opened on the unit specified as first parameter. The point, which is supplied as second parameter, must have the same format as the value of one of the record-input-output procedures inrec, outrec, invar or outvar, i.e.

<blocknumber> .19 + <blockrelative> .39

The effect of the look-up-process will depend on whether the file is open for input or output:

Input

The tape is positioned and the pointers in the file table adjusted so that a following activation of one of the procedures inrec or invar will have as value the same file point as is specified as parameter to the call of research, i.e. it will give access to the same record as the call did, which supplied the file point originally.

Output

If the filepoint is specified beyond that of the last record output on the file, the effect is undefined. If it describes a record at a point before the last record, the tape will be positioned and the pointers in the file table adjusted so that a following activation of one of the procedures outrec or outvar will have as value the same file point as is specified as parameter to the call of research.

Value — The value of the procedure indicates to the user the result of the look-up-process in the following way:
Research = 0 => The look-up was successfully with the effect as described above.
Research ≠ 0 => The specified point could not be found, or tape-errors have occurred when the ultimate block was read into the double buffer area.

6.3.12

integer procedure close (unit, labelarray);

Legality — Tapestatus: Allowed for openin or openout
Filestatus: Open for input or open for output

Function — The function depends on whether the file has been opened for input or for output:

Input

The filelabel following the file to be closed is read from the tapeunit specified as first parameter. If close is called before the EOF-reaction (see section 7 and description of inrec), this will be preceded by a search forward until the file mark has been read.
The words of the user accessible part of the label (see section 3) are transferred to the first (max. 8) words of the array specified as second parameter. Finally, the buffer area - which was reserved by the corresponding open call - is released (cf. section 5).

Output

The last activated block transport is checked, and a possible not completed last block of the file is written on the unit specified as first parameter and checked. If any tape-errors occur during these operations, the normal error recovery mechanism (as described in section 7) will be activated. After the output-buffer has been emptied, a file mark and a dummy file label - which indicates the last file on the tape - are written on the tape. The file label will have the normal format (cf. section 3.2.) and will include the first (max. 8) words of the array specified as second parameter. Finally, the reserved buffer area will be released.

Value — The value of close indicates to the user the result of the call in the following way:

close > 0 => The file has been closed, and the value is the number of free words in the buffer store.

close = 0 => The file was already closed, and the call has been dummy.

close < 0 => The file has been closed, but it has been impossible to read or write the following file label.

Filestatus — Closed.

6.3.13

integer procedure torec (name, fromarray, index);

Legality — Tapestatus: Irrelevant
Filestatus: Irrelevant

Function — The procedure will transfer a number of words from the array specified as second parameter to the elements of a record [x] identified by the boolean variable specified as first parameter. The integer third parameter specifies the number of the first element inside the array [xx] to be transferred (index = 1 specifies the first element of the array). In this way it is possible to move elements from an array independent of its structure.

Value — The value of the procedure is the number of words transferred.

Warning — See warning in description of procedure record (6.3.5).

x) Cf. section 4.

xx) Interpreted as having one dimension only.

6.3.14

integer procedure fromrec (name, toarray, index);

Legality — Tapestatus: Irrelevant
Filestatus: Irrelevant

Function — The procedure will transfer information from the elements of a record [x] identified by the boolean variable specified as first parameter to a corresponding number of elements in the array specified as second parameter. The integer third parameter specifies the number of the first element in the array [xx] to receive information (index = 1 specifies the first element of the array). In this way it is possible to move information to an array independent of its structure.

Value — The value of the procedure is the number of words transferred.

Warning — See warning in description of procedure record (6.3.5).

[x] Cf. section 4.

[xx] Interpreted as having one dimension only.

6.3.15

integer procedure blockin (unit, dataarray, mode);

Legality — Tapestatus: Allowed for blockin
Filestatus: Closed

Function — The reading of a block from the tapeunit specified as first parameter to the array specified as third parameter is activated. The reading will be performed according to the value of the second parameter in the following way:

mode

0 : 7 characters/word, even parity
1 : 7 - , uneven parity
2 : 4 - [x] , even parity
3 : 4 - , uneven parity.

The procedure will return as soon as the transport is activated, and no checking will take place. If the block happens to be longer than the array, the remaining words of the block will not be transferred to the buffer.

Value — The value of blockin will normally be zero, but if the current tape is positioned at load point, a test is activated if it is the correct tape, i.e. the tape that was mounted at the time of the last call of tapetest.
If this test shows any difference, the value of tapetest will be either -1 or 1, the negative value indicating that tape errors - occurring when reading the tape label - may have caused the difference.

Note: — No tape operations will have been activated, if the procedure returns with a non-zero-value.

x) Cf. ref. 1, page 2.

6.3.16

integer procedure blockout (unit, length, dataarray, mode);

Legality — Tapestatus: Allowed for blockout
Filestatus: Closed

Function — The writing of a block from the array specified as third parameter to the tapeunit specified as first parameter and with length as specified in the second parameter is activated. The writing will be performed according to the value of the fourth parameter in the following way:

mode			
0 or 4:	7 characters/word,		even parity
1 or 5:	7	-	, uneven parity
2 or 6:	4	- x)	, even parity
3 or 7:	4	-	, uneven parity.

The values 4-7 imply a writing after erasure of about 12 cm tape. The values 0-3 imply a normal writing. The procedure will return as soon as the transport is activated, and no checking will take place.

Value — The value of the procedure is exactly as described for procedure blockin.

x) Cf. ref. 1, page 2.

6.3.17

integer procedure status (unit, length);

Legality — Tapestatus: Allowed for blockin or for blockout
Filestatus: Closed

Function — The status word of the last activated blocktransport on the tape unit specified as first parameter is processed so that the actual length of the block transferred will be available in the integer variable specified as second parameter.

Value — The value of the procedure is the first four bits (interpreted as an integer) of the above mentioned status word.
The meaning of these bits is defined in ref. 1, page 3.

6.3.18

procedure — rewind (unit);

Legality — Tapestatus: Irrelevant
Filestatus: Closed

Function — The tape on the unit specified as parameter is rewound to loadpoint. The procedure will return as soon as the operation is activated.

6.3.19

procedure — unload (unit);

Legality — Tapestatus: Irrelevant
Filestatus: Closed

Function — The tape on the unit specified as parameter is rewound to loadpoint, and the unit is set to local control.
The procedure will return as soon as the operation is activated.

6.3.20

boolean procedure drive (unit, mode);

Legality — Tapestatus: Allowed for blockin or blockout
Filestatus: Closed

Function — The procedure will activate a tape operation on the unit specified as first parameter according to the value of the second parameter and return immediately.

mode = 0
The operation is equivalent to the sense busy instruction.

mode = 1 or mode = -1
The procedure activates a skip block forward or a skip block backward, respectively.

mode = 2 or mode = -2
The procedure activates a skip file forward or a skip file backward, respectively.

Value — The value will always be false except when mode = 0 and the unit is busy. In this case the value is true.

7. Treatment of errors, end-of-file and end-of-tape

7.1. Formal errors

It is a wellknown fact that it is nearly impossible in practice to make programs correct at the first time. Before a program can be used for routine runs, a number of errors must be removed from it.

These errors may be either formal errors - i.e. incorrect use of the elements of the programming language - or real programming errors which cause the program to react in another way than it was intended to. The discussion of programming errors will, however, be beyond the scope of this manual.

Formal errors in connection with SAMBA may be classified as:

a: Static formal errors.

b: Dynamic formal errors.

Static formal errors are present whenever a SAMBA procedure is used in an illegal way with respect to kind and type in a statement, when it is supplied with a wrong number of parameters, or when these parameters are of wrong kind or type. Such errors are all catched during the translation and are treated like normal translation errors by the GIER ALGOL 4 compiler.

Dynamic formal errors occur when a SAMBA procedure is called in a situation where its activation is illegal with respect to tapestatus or filestatus as described in section 6.2. Moreover whenever a wrong value is supplied as parameter, e.g. by specifying the opening of a non-existing file on a given unit. These errors will normally be recognized during the run of the program, since they are caused by use - or more correct by misuse - of the system. It will cause the run to be terminated by an error message. The format of this message will by the same as other running errors recognized by GIER ALGOL 4 (since they are treated by the same procedure in the running system). A complete list of the error messages activated by SAMBA and their possible cause is given in appendix B.

7.2. Tape errors

When all formal errors as well as programming errors have been removed (or at least seem to have been removed), a user dealing with magnetic tapes and with hardware for handling magnetic tapes will still have to live with the so-called tape errors.

This fact has been paid much attention to during the design of SAMBA, and it has therefore been possible to make the detection and handling of tape errors fully integrated with all other functions of the system.

However, the consequence of tape errors both to the system and to the user is very much dependent on the level in the use of the system on which the error appears.

Tape errors recognized when tape labels or file labels are processed will normally be rather fatal for the following process. When a procedure - which is dependent on the information contained in these labels - cannot remove an error by an internal recovery mechanique, this is announced for the user by a negative value of the procedure (see description of the procedures tapetest, getfile, openin, openout and close).

When tape errors occur during input or output of records, the situation is normally quite another. The remaining part of this section will deal with these problems only.

7.2.1 Philosophy

Before the handling of tape errors in the record processing part of SAMBA is described in detail, a few words must be said about the basic philosophy which lays behind the implementation.

The following points are in no way intended to give the full set of reasons why the actual solution has been chosen, but it may help the user to understand it, and thereby make it easier to take up the idea if the system is going to be used in future programs.

Nearly all programs dealing with great amounts of uniform structured data (file processing) must accept errors of one kind or another in these data, like for instance punch errors or control digit errors in raw input, illegal transactions to an existing or non-existing master record in an

updating process and so on. It is a very primitive program which will stop running when such errors are encountered, and in practice it is just inacceptable to give up the process. It may very often even be a main task of a program to recognize all errors and to document them for later correction, all in parallel to processing the correct data.

While this principle - the process must go on - is commonly accepted when the talk is about the above mentioned kind of errors, most users seem to be nearly paralysed when they have to take action upon tape errors. Often common practice is here simply to run the whole program or set of programs once more from the very beginning or even run it with an earlier generation of the files involved in order to get rid of a simple tape error.

This ineffective practice, however, is very often caused by use of bad or primitive tape systems, of which many even cannot detect all errors, not to mention that they do not allow the user to react in a reasonable way on them.

The appearance of a tape error should be considered a normal event, and a tape handling system should allow the user to treat a tape error by an action from the program like treatment of any other data error and in an easy way make it possible to document what is wrong and to continue the process with a clear and simple logic.

7.2.2 Detection and recovery

Parity and blocklength errors

The input-output of records in SAMBA is treated by a double buffer administration which will activate the necessary block transports and check them according to the information derived from the statusword of the tape unit and from the second word of the blocklabel (cf. ref. 1 and section 3.3.). Hereby it is possible to detect all parity and blocklength errors.

When an error of this kind is recognized, the system will automatically try to recover it by backspacing the current block and try the read/write operation once more [x]. This will be repeated up to three times before the system will give up and leave it to the user to take the proper action as described below (section 7.2.3).

[x] In this case the write operation will start with erasure of about 12 cm tape in order to try to get free of a possible bad spot on the tape.

Each time a reread/rewrite operation is activated on a given unit in the recovery process, a special counter assigned for the unit is counted up by one. These counters are initialized to zero in the init-run-process of GIER ALGOL 4, but their contents will survive the return to the (HELP-) program which activated the run and may be used or collected for maintenance purpose.

The counters are located in consecutive words in the core store so that the counter for unit 1 has the address c17+8, where c17 is the SLIP-address of UV (cf. ref. 2).

Blocknumber errors

If no parity error or blocklength error is encountered, or if they have been removed by the recovery, the double buffer administration will continue with a test on proper block sequence. This is done by means of the blocknumber stored in the first word of the blocklabel (cf. section 3.3.) which is compared to an internal block counter stored in the filetable and counted by one each time a new block is read or written.

If a block read in from a tape is found to be out of sequence, this can either be due to a parity error which may have destroyed the blocknumber in the blocklabel, or it may indicate that the tape has not been generated in a correct way (i.e. by using the proper SAMBA-procedures). In any case, it has no meaning for the system to try to recover this special error, and it is up to the user to decide what to do, as described below (section 7.2.3).

Checksum errors

When an input block has been read without any error of the kinds mentioned above (or when an erroneous block has been accepted as described below), the records of the block will be used one by one by the procedures inrec or invar. If these records were provided with a checksum when they were output and if it was specified in the call of openin, each record will be tested against its checksum (cf. description of procedures inrec and invar).

If a checksum error is found, no recovery will take place, and the system will immediately leave it to the user to decide the action.

7.2.3 Activation of the interrupt procedure

When a tape error has been recognized by the system and a possible recovery has turned out to be fruitless, the system will give up. SAMBA will now activate a user declared ALGOL-procedure, which serves as a kind of interrupt action and must be specified to the system by means of the procedure setproc after the current file has been opened (see section 6.3.10).

If no such procedure has been specified, the error will be ignored as described in 7.2.4.

The interrupt procedure must be declared with a head as shown in the following example:

```
procedure error (unit, type, area, length);
value unit, length; integer unit, type, length; array area;
```

When the procedure is activated by SAMBA, the parameters will contain the following information about the actual situation:

unit: supplies the value of the current unit

type: specifies the kind of error by the following values:
1 = checksumerror
2 = parity error x)
3 = blocklength error
4 = blocksequence error

area: describes the erroneous field as if the system had declared a single dimension array with element No. 1 in the first word of the field. If type = 1, area will cover an erroneous record including the checksumword. If type > 1, area will cover an erroneous block including the two block-label-words (cf. section 3.3.)

x) If a parity error - which occurs when a block is read - cannot be removed by the recovery process and if a recordsum is available and is checked by the system, the interrupt procedure will not be activated before a possible checksum error occurs. In this way, errors - which have only destroyed the parity bits, but not the real data - will not be indicated for the user.
Note: This is only valid for parity errors.

length: specifies the length of the above described array area, i.e. the length of the bad record or block.

These parameters will contain enough information to make it possible inside the error procedure to document the error in a convenient way. If more advanced actions are wanted, it may be necessary to make use of information in variables which are global to the interrupt procedure, i.e. variables which are declared and updated in the main program.

The call of the interrupt procedure is performed in a fully ALGOL correct manner, which means that the user is allowed to make use of all normal ALGOL possibilities inside the procedure, including call of SAMBA procedures which may be relevant if the user has programmed a more advanced recovery procedure (e.g. by means of research), or if one or more files must be closed or opened (cf. section 7.3.).

Warning The user must pay attention to the fact that no test is performed whether or not the ALGOL-rules concerning scope are sacrified. If a record input/output-procedure (in some strange programs) is called at a blocklevel outside the scope of a specified interrupt-procedure, the program will go completely off its tracks if the system activates the interrupt procedure.

Moreover, the user is warned against any seriously intended attempts to activate the record input/output procedures on a given unit recursively. The effect of such calls (which may happen again to activate the interrupt procedure etc.) may easily turn out to be completely undefined.

7.2.4 Exit from the interrupt procedure

After proper treatment of the error, control may be given back to SAMBA by leaving the interrupt procedure through the end.

However, before returning, the user must specify to the system how it is going to proceed. This is done by means of the second parameter - type - which may be changed or not ×). The exact reaction of the system hereupon

×) In order to make the changing have an effect, this parameter must of course not be value specified.

will depend both on the original value of type and on the error having been caused by an input or output operation. There are three possibilities to specify a reaction:

a) Type is unchanged

The main effect is that the error is ignored and - by the way - is equivalent to the situation that no interrupt procedure is specified. To ignore the error, simply means that the system will continue the process with the bad area accepted xx). However, before returning to the program, the contents of the boolean standard variable sambaerror will be changed to contain the following information:

<1>.0 + <unit>.5 + <type>.9 + <blocknumber>.29 + <blockrelative>.39

This standard variable is initialized to 40 0 when the run is started and will only be changed by the system in situations as described above.

b) Type = 0

This is equivalent to asking the system to try once more. The action here will primarily depend on the original error type:

If it was a checksumerror, the system will return to the entry of the record input procedure and try to get the same record once more. This will have no meaning for the user, since the system will just redetect the checksumerror, activate the interrupt procedure again and thereby probably go into an infinite loop.

If originally the error was of type 2, 3 or 4, the system will reactivate the internal recovery process with the consequenses as described in section 7.2.2.

c) Type < 0

This is accepted as an instruction to the system to skip the bad area. If the error is caused by an output operation, the unit will be backspaced over the bad area.

xx) If the error was a blocksequence error, the contents of the blocknumber word of the blocklabel in the wrong block will be accepted as the current blocknumber.

If the error is a checksumerror, the system will continue with the next record in the file and leave the bad one untransferred.

If the error is a reading error or a blocksequence error, the bad block will be left untouched. Instead, the internal blocknumber is counted up by one, and the next block is read in and checked as described in section 7.2.2.

7.3. End-of-file and end-of-tape

As described above, each blocktransport will be checked in different ways by the double buffer administration. This includes a test on the two bits in the status word which indicates end-of-file read (EOF) and end-of-tape sensed (EOT) ×).

EOF in SAMBA indicates that no more records is available on a given file. This must be communicated to the user so that he can take the proper action which will normally include a call of the procedure close in order to release the occupied buffer area and make the tape-unit available for possible further processing of other files.

EOT indicates (independent of system) that the tape is very near to its end (probably only a few meter remains). If it is sensed during output of a block, SAMBA must also communicate this to the user, since the system does not include an automatic administration for multi-tape processing. The user himself has to program such an administration, which must include a call of close for the current output file and a re-opening of it on another tape. (A proposal for the logic of a multi-tape-administration using SAMBA is given in appendix C).

Since both situations in most programs with a clear and straightforward logic must be irrelevant to the detailed record processing, SAMBA will treat them as a kind of pseudo tape errors, which means that they will cause the interrupt procedure to be activated.

This gives the user the same freedom to take any proper action as for ordinary tape errors but will not include means for specifying alternate actions by the system upon return to it, since only one action is considered relevant.

When the interrupt procedure is activated, the first parameter will specify the value of the unit as before while the second parameter will be equal to 5 in case of EOF and equal to 6 in case of EOT. The last two parameters are irrelevant.

×) Cf. ref. 1, page 3.

Before the interrupt procedure is activated, the internal representation of filestatus (cf. section 6.2.) is changed in order to make any further input/output operation illegal on the current file. However, the user himself has to call the procedure close before the occupied buffer area can be released and the tape unit made available for possible processing of other files. This call of close may conveniently be performed inside the interrupt procedure.

Upon return from the interrupt procedure, the system will return directly to the main program with the value of the record-input-output-procedure equal to 40 0. This makes it possible for the user to take care of the situation from the main program, if this is found convenient or if no interrupt procedure is specified.

In any case, the effect of the call, which caused the EOF/EOT situation to be detected, will be equivalent to a dummy call (apart from the effects mentioned above). This means that the elements of the specified record (for inrec) or array (for invar) have not been changed and that the specified record (for outrec) or part of an array (for outvar) have not been output.

7.4. End-of-tape combined with tape errors

If EOT is sensed after a write block operation together with an error of type 2 or 3, the system will just store the information about the EOT and activate the automatic recovery process. If the error can be removed now, the system will just recognize it as a normal EOT-situation.

If on the other hand, the error cannot be removed by the recovery process, a special situation has arisen. This is announced to the user by a call of the interrupt procedure with type = 7. The other parameters contain information as described in section 7.2.3. However, since EOT has been sensed, the user will have no possibility to specify alternative reactions upon return to the system.

The system will return to the main program with the value of the record-output-procedure equal to 40 0 and with the standard variable sambaerror set according to the situation. Besides this, the internal representation of filestatus has been changed in order to avoid more output on the current file.

Appendix A:

SAMBA tape format

The format and contents of a standard SAMBA tape may be described by the following Baccus notation scheme:

<SAMBA tape> ::= <SAMBA label><FM><data files><end label>

<SAMBA label> ::= GIER <tapeno><tape pattern>

<tape no> ::= <four decimal digits>

<tape pattern> ::= <40 bits>

<FM> ::= <file mark as defined in ref. 1, page 3>

<data files> ::= <empty>|<datafile>|<datafiles><datafile>

<data file> ::= <file label><data blocks><FM>

<file label> ::= <systempart><userpart>

<system part> ::= <mode.9 + area.25 + filenumber.39><labellength.19>

<user part> ::= <max eight 7-character words>

<data blocks> ::= <empty>|<data block>|<data blocks><data block>

<data block> ::= <block label><records>

<blocklabel> ::= <blockno.19><blocklength.19>

<records> ::= <record>|<records><record>

<record> ::= <checksumword><data words>

<checksumword> ::= <empty>|<recordlength.9><recordsum>

<recordsum> ::= <empty>|<compressed sum of recordwords.39>

<data words> ::= <GIER word>|<data words><GIER word>

<GIER word> ::= <4 or 7 characters in uneven parity>

<end label> ::= <file label>

Appendix B:

Error messages

As mentioned in section 7.1, dynamic formal errors will cause the run to be terminated with an error message. This message will be generated by the standard error routine in the running system of GIER ALGOL 4. However, the format has been changed slightly to specify the value of the unit-parameter in the last activated call of a SAMBA procedure before the error was recognized. The format of the error message will therefore be (cf. ref. 2, appendix 2):

<text><unit><line 1>-<line 2>,<rel.track>

The possible texts and the situation causing them are:

error 14 A SAMBA procedure which is illegal with respect to the current value of tapetest has been activated (cf. section 6.2.).

error 15 A procedure which requires the corresponding file to be open has been activated, when filestatus indicates a close situation.

error 16 A procedure which requires a close situation has been activated, when filestatus indicates a file to be open.

error 17 A parameter to a SAMBA procedure has a value, which exceeds the legal range.

error 18 An attempt to look up a file which number is greater than the number of relevant files on the specified tape.

error 19 The size of one record to be output is greater than the specified maximum block size for the file, or the total size of the records required to be input does not match the size of the block to which they belong.

buffer A SAMBA procedure requires more area in the buffer than is available, or an array has been declared which is greater than the remaining free area in the buffer. NOTE: This error message will replace the message array from the normal version of GIER ALGOL 4.

Appendix C

This appendix contains a program and two procedures programmed in GIER ALGOL 4, which illustrate some of the features of SAMBA and how the system may be used.

The first example describes a possible solution of a simple file updating problem involving one master file with fixed length records and one transaction file with variable length records.

The second example shows the declaration of a possible interrupt procedure,
ibes in principle how a simple system for
ay be programmed.

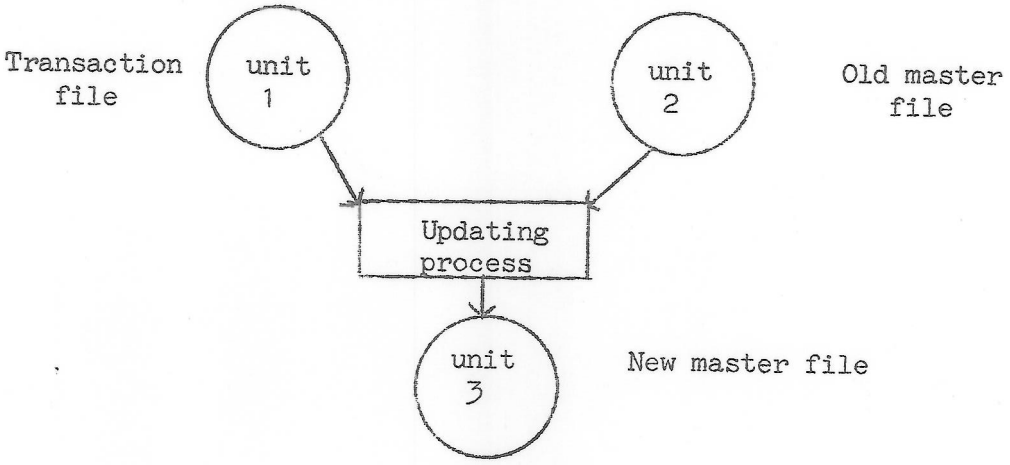
examples 2 and 3 are in fact very primitive
ufficient for practical purpose. However, by
procedures, e.g. by assuming that more infor-
nd from the main program by means of global
ough introduction of adequate conventions for
rt of the file labels, it should be possible
r system, which may be common to and convenient
obtained hereby would be standardization of
r reactions.

LGOL 4 will make it easy for a user to incor-
named area on drum, disk or from a system tape

Example 1

This example is in no way intended to describe the solution of a real ADP-problem, but it contains most of the elements which are common in normal file updating problems and describes how easy SAMBA fits the logic of the solution.

Consider a trade company which has the necessary information about its customers stored as records on a master file. In a merge process, this master file is updated by means of information from another file consisting of so-called transaction-records, which have been generated in a previous input and sorting process.



A record on the master file contains the following information:

Word No.	Contents	Corresponding ALGOL-variable
1.	A number representing the customer:	integer m
2.	A number representing the day the customer was introduced to the system:	integer start date
3.	A number representing the day of the last processed transaction to the master record:	integer last date
4.	The amount of the last transaction to the master record:	integer last amount
5.	The total amount which the customer owes to the trade firm:	integer total amount
6-10.	Five words containing a text representing the name and address of the customer:	boolean array NAME [1:5]

The records on the master file will be processed with checksum.

A record on the transaction file may have three different formats but the two first words will always contain:

1. A number representing the customer: integer t
2. A number indicating the type of transaction as described in the following: integer type

type = 1: The transaction introduces a new customer to the system. The remaining words of the transaction record are in such a case:

3. A number representing the transaction day: integer date
4. A number representing a possible amount to be registered: integer amount
5-9. Five words containing a text representing the name and address of the customer: (no explicit area)

type = 2: The transaction announces a change in the amount owed. In this case the transaction will only consist of the words 1-4 described above.

type = 3: The transaction announces that a customer may be removed from the system if the total amount is equal to zero. In this case only the two first words mentioned above will form the transaction.

The transaction will contain no extra word for checksum.

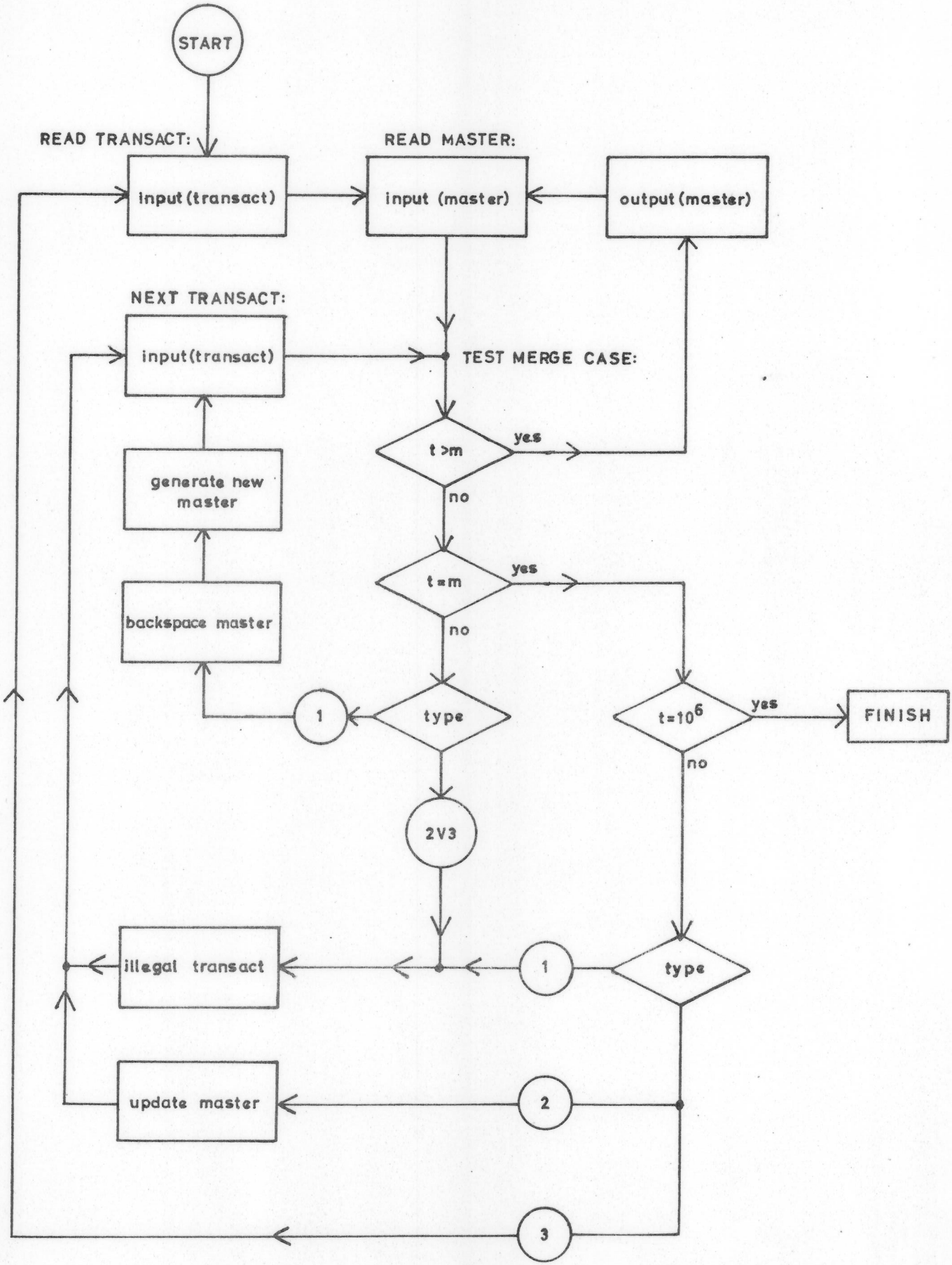
The records of the two files are both ordered with ascending numbers in word 1, and the transactions are moreover ordered with respect to type so that type-1-records appear before type-2- and type-3-records with the same number.

Both files are written in high density mode on the tape.

The main logic of the process is described in the following scheme:

	t < m	t = m	t > m
type = 1	generate a new master record	illegal transaction (master already present)	transfer master to output
type = 2	illegal transaction (no master to update)	change master	transfer master to output
type = 3	illegal transaction (no master to delete)	delete master and input new master	transfer master to output
	and input new transaction		and input new master

The action in the interrupt procedure upon EOF on one of the two input files is just to set the respective number of the record to a certain value greater than the greatest possible number in the system. Here 1000000 has been chosen.


START
READ TRANSACT:
READ MASTER:
input(transact)
input (master)
output(master)
NEXT TRANSACT:
input(transact)
TEST MERGE CASE:
t >m
yes
no
generate new master
t =m
yes
no
backspace master
1
type
t=10^6
yes
FINISH
no
2V3
illegal transact
1
type
update master
2
3

Updating program (example 1)

```
begin integer m, start date, last date, last amount, total amount,
        t, type, date, amount, tapeno, unit, value;

  boolean array NAME [1:5];
  integer array TRANSACT [1:9], LABEL [1:8];
  boolean ident2, master, transact, masterpoint;

  procedure tapetesterror (unit, tapeno, value); integer unit, tapeno, value;
    begin comment action upon error in tape identification process end;

  procedure opencloseerror (unit, OPEN); integer unit; boolean OPEN;
    begin comment action upon open/close errors end;

  procedure interrupt (unit, type, area, length); integer ..... etc;
    begin comment see example 2 for possible procedure body end;

START: select (17); writetext ({<tape identification in reader }); lyn;
    select (16); comment now follows tape identification process;
    for unit:= 1,2,3 do
      begin tapeno:= readinteger; ident2:= boolean readinteger;
        value:= tapetest (unit, tapeno, ident2, if unit ≠ 3 then 1 else 3,0);
        if value ≠ 0 then tapetesterror (unit, tapeno, value)
      end;

OPENTRANSACTION:
    comment first file on unit 1 without checksum;
    if openin (1,1,LABEL,0) < 0 then opencloseerror (1,true);
    comment here may follow a possible processing of the contents of LABEL;

OPENOLDMASTER:
    comment first file on unit 2, with checksumcontrol during input;
    if openin (2,1,LABEL,1) < 0 then opencloseerror (2,true);
    comment here a possible processing of the contents of LABEL may follow;

OPENNEWMASTER:
    comment first file on unit 3, reserve 1000 words for double buffer,
      generate checksum and store in each record during output;
    comment here the array LABEL may be initialized to contain user
      information for the file label;
    if openout (3,1,LABEL,1000,3) < 0 then opencloseerror (3,true);

    setproc (1,interrupt); setproc (2,interrupt); setproc (3,interrupt);

DECLARE RECORDS:
    record (master, m, start date, last date, last amount, total amount, NAME);

    record (transact, t, type, date, amount);
    record (newname, NAME);

READ TRANSACT: invar (1,TRANSACT,value); torec(transact,TRANSACT,1);
    comment now the variables t, type and maybe date and amount will
      contain relevant information about the transaction. The contents
      of TRANSACT [5:9] will only be moved when a new master must be
      generated. If EOF was read, t will be equal to 1000000 due to
      action in the interrupt procedure;
```

```
READ MASTER: masterpoint:= inrec (2,master)
    comment now the current master has been read and moved to the
      variables: m, start date, last date, last amount, total amount
      and NAME. If EOF was read, m will be equal to 1000000 due to
      action in the interrupt procedure;

TEST MERGE CASE:
    if t > m then
      begin outrec (3,master); goto READ MASTER end;

    if t = 0 then
      begin if t = 1000000 then goto FINISH;
        comment EOF has been read on both input files;
        goto case type of ILLEGAL, UPDATE, READ TRANSACT;
      end;

    comment code for t < m follows here;

    if type = 1 then
      begin comment introduce new master;
        research (2,masterpoint);
        comment now a following inrec-operation on unit 2 will give the
          current master once more, and the master record may be used
          for the new one;
        m:= t;
        startdate:= lastdate:= date;
        lastamount:= totalamount:= amount;
        torec (newname,TRANSACT,5);
        comment this call moves the name-text from the transaction area
          to the array NAME of the master record. Now the generation of
          the new master has been accomplished;
      end
    else
ILLEGAL:
      begin comment action upon an illegal transaction end;

NEXT TRANSACT:
    invar (1,TRANSACT,value); torec(transact,TRANSACT,1);
    comment see remarks under READ TRANSACT;
    goto TEST MERGE CASE;

UPDATE:
    comment actions when t = m and type = 2;
    last date:= date;
    last amount:= amount;
    total amount:= total amount + amount;
    goto NEXT TRANSACT;

FINISH:
    comment action when EOF has been read on both input files;
    close (1,LABEL); close (2,LABEL); unload (1); unload (2);
    comment not necessary since no more process will take place.
      If any information is to be stored in the dummy file label
      following the new master, the contents must be stored in the
      array LABEL now;
    if close (3,LABEL) < 0 then opencloseerror (3,false); unload (3);

    select (17); writetext (‡< FINISH ‡)
end example 1;
```

Example 2:

```
integer procedure interrupt (unit, type, area, length);
  integer unit, type, length; integer array area;
  begin integer medium, j;
    medium:= select (if type > 4 then logmedium else errormedium),
    writecr;
    writetext ({<unit}); writeinteger ({dd}, unit); writechar (0);
    writetext (case type of ({<sumerror}, {<parityerror},
               {<blocklengtherror}, {blocknoerror}, {<EOF read},
               {<EOT sensed}, {<EOT sensed}));

    if type < 5 then
ERROR:begin for j:= 1 step 1 until length do
        begin writecr; writeinteger ({dddd}, j);
          writeinteger ({ddddddddddd}, area [j])
        end;
      type:= -1
      comment specify 'skip bad area upon return';
      end type < 5
    else
      begin if type = 5 then
EOF:    begin integer infinite;
          comment only single reel input files considered;
          infinite:= number greater then greatest criterium value;
          if unit = 1 then criterium 1:= infinite
            else criterium 2:= infinite
        end
      else
EOT:    begin comment separate action on error + EOT not relevant
          here since all bad blocks will be skipped;
          next reel (unit);
          comment see possible declaration in example 3
        end EOT
      end type ≥ 5;
    select (medium)
  end interrupt procedure;
```

Example 3:

The following procedure which may be used both for input and output files will assume basic information about the relevant files stored in an integer array FILE [1:maxunit, 1:4]; the contents of each row is:

column:	contents:
1	if output then 0 else 1
2	reel sequence number
3	if output then area else irrelevant
4	if output then mode else check

```
. . . . . .                                   . . . . . .
procedure next reel (unit); integer unit;
  begin integer ident1, value; boolean ident2, output;
    output:= FILE [unit, 1] = 0;
    if output then prepare EOT label;
    if close (unit, LABEL) < 0 then opencloseerror (unit, false); x)
    if -, output then process information from EOT label;
    FILE [unit, 2]:= FILE [unit, 2] + 1; unload (unit);
    writetext ({<mount reel sequence No.});
    writeinteger ({ddd}, FILE [unit, 2]);
    writetext ({<on unit}); writeinteger ({dd}, unit);
    get tape ident (unit, ident1, ident2);
    comment this call will supply the information necessary to identify
      the next reel;
    value:= tapetest (unit, ident1, ident2, if output then 3 else 1, 0);
    if value ≠ 0 then tapetesterror (unit, ident1, value);          x)
    if output then
      begin prepare start label;
        if openout (unit, 1, LABEL, FILE [unit, 3], FILE [unit, 4]) < 0
          then opencloseerror (unit, true)                          x)
      end
    else
      begin
        if openin (unit, 1, LABEL, FILE [unit, 4])< 0
          then opencloseerror (unit, true);                         x)
        process information from start label
      end
end procedure next reel;
```

x) see comments in example 1.

Appendix D:

Standard values of tapestatus

The rules in the standard version of the SAMBA compiler which governs the result of a call of tapetest are based upon the following philosophy:

a) Output on a given tape is only allowed if it has been possible to identify the tapelabel fully.

b) If the relevant fields of a tapelabel cannot be identified, but a tape error has been recognized, input from the tape will be allowed.

c) If the tape is recognized as a SAMBA tape - i.e. field 1 contains the characters GIER - only the proper SAMBA input/output procedures will be allowed.

d) If the tape is recognized as a working tape, both the proper SAMBA procedures and the basic tape procedures will be allowed.

e) If the tape is recognized as non-standard tape, only basic tape procedures will be allowed.

f) No proper SAMBA procedures will be allowed, if the rewind parameter to tapetest is one.

The table below shows the exact values of tapestatus in the relevant cases as implemented in the standard version. If some installation should want to change the values (and thereby the rules), the procedure described in appendix E should be followed.

A '+' in the left columns means yes to the condition, while '-' means no. A 'one' in the four right columns means that the procedure will be allowed - a 'zero' that it will not be allowed.

tape class	ident 1 ok	ident 2 ok	rewind = 1	tape-error	value of tapetest	in/out rec	block in/out
	-	-	-	-	3	0 0	0 0
	-	-	-	+	-3	0 0	0 0
	-	-	+	-	3	0 0	0 0
	-	-	+	+	-3	0 0	0 0
	-	+	-	-	2	0 0	0 0
	-	+	-	+	-2	1 0	0 0
	-	+	+	-	2	0 0	0 0
	-	+	+	+	-2	0 0	0 0
gier	+	-	-	-	1	0 0	0 0
	+	-	-	+	-1	1 0	0 0
	+	-	+	-	1	0 0	0 0
	+	-	+	+	-1	0 0	0 0
	+	+	-	-	0	1 1	0 0
	+	+	-	+	0	1 1	0 0
	+	+	+	-	0	0 0	0 0
	+	+	+	+	0	0 0	0 0
	-	+	-	-	1	0 0	0 0
	-	+	-	+	-1	1 0	1 0
	-	+	+	-	1	0 0	0 0
rcar	-	+	+	+	-1	0 0	1 0
	+	+	-	-	0	1 1	1 1
	+	+	-	+	0	1 1	1 1
	+	+	+	-	0	0 0	1 1
	+	+	+	+	0	0 0	1 1
	-	-	-	-	7	0 0	1 0
	-	-	-	+	-7	1 0	1 0
	-	-	+	-	7	0 0	1 0
	-	-	+	+	-7	0 0	1 0
	-	+	-	-	6	0 0	1 0
non	-	+	-	+	-6	1 0	1 0
	-	+	+	-	6	0 0	1 0
standard	-	+	+	+	-6	0 0	1 0
	+	-	-	-	5	0 0	1 0
	+	-	-	+	-5	1 0	1 0
	+	-	+	-	5	0 0	1 0
	+	-	+	+	-5	0 0	1 0
	+	+	-	-	4	0 0	1 1
	+	+	-	+	-4	1 0	1 1
	+	+	+	-	4	0 0	1 1
	+	+	+	+	-4	0 0	1 1

Appendix E:

Generation of a GIER ALGOL 4/SAMBA compiler

The generation of the SAMBA version of the GIER ALGOL 4 compiler will follow exactly the same rules as specified in ref. 2, section 14.2 with the following additions:

During loading of the T1,L1-tape, GIER will stop with the message:

redefine wanted loading parameters.

In this situation the name e40 must be set equal to the number of tape units available on the installation. When the loading is restarted, the SAMBA-version of the compiler will be generated.

After loading the T7,L3-tape, the following message will occur:

load SAMBA-tape.

Now the tape marked 'SAMBA' must be loaded. During this GIER will stop with the message:

redefine SAMBA system.

Now loading may be restarted directly or some internal loading parameters may be changed as described below.

The SAMBA-tape is just a simple extention of the normal library tape, and it will therefore terminate with the message T7,L3. Hereafter the loading process will proceed as for any other compiler.

Special SAMBA-compilers

Since many of the standard procedures of SAMBA will use part of the code of each other (often in a very complicated manner), it has been convenient to treat the procedures as one single procedure with many entries. The consequence of this, however, is that all the SAMBA-procedures will be included in the translated program, even if only one procedure is used.

In some situations, however, it may be of interest only to use part of the system and have a compiler which only includes the necessary code. This has been taken into account so that it is possible to generate three special versions of the SAMBA-compiler which contain SAMBA-procedures as shown in the following scheme:

Name of procedures	No. of tracks occupied	Version: A	B	C
tapetest	2	×	×	
openin/openout/getfile	4	×		
close	1	×		
record	3	×		×
inrec/outrec	1	×		
invar/outvar	1	×		
torec/fromrec	1	×		×
research	2	×		
rewind/unload	1	×	×	
blockin/blockout	1		×	
drive/status	1		×	
setproc/blockchange x)	5	×		
release xx)	1	×		×

The respective versions will be generated automatically, if the following definitions of SLIP-names are performed:

Version A: Following message 'redefine SAMBA-system' : b31 = 1

Version B: - - 'redefine SAMBA-system' : b32 = 1

Version C: - - 'redefine wanted loading parameters' : e40 = -1

Note that version C may be used even when no tape units are available.

It should be mentioned that the system contains facilities to generate more specialized versions than those specified above. Requests for instructions to follow in such a case should be directed to GIER SYSTEM LIBRARY.

Changes of rules for evaluation of tape status

As mentioned in the description of procedure tapetest, the rules for evaluating tapestatus are stored in a table available for tapetest.

This table may be changed from the standard contents shown in appendix D during the loading of the SAMBA tape.

Following the message 'redefine SAMBA system', the SLIP-name b30 must be set equal to one. This will cause GIER to stop with the following message when the procedure tapetest is loaded:

read tapetest redefinition tape.

x) Used internally for double buffer and tape error administration.

xx) Used internally for administration of available buffer store area.

Now a new table can be read in. This table must have been generated in a previous process by the program 'Generate SAMBA tapetest-table' which is available through GIER SYSTEM LIBRARY. The system will check that a correct tape is used, otherwise the message 'SAMBA alas' will be typed out, and the reading will stop with no possibility to restart.

When the redefinition tape has been read in, the loading process must proceed with the remaining part of the SAMBA tape as described above.

Extension of the record transfer area

As mentioned in the description of procedure record, transfers to and from elements of a declared record will take place in groups of consecutively stored words.

However, when words are transferred from one part of the buffer store to another, which is the case for elements of arrays, the transfer must be performed via a working area in the running system part of the core store. In the standard version of the SAMBA-compiler, this working area consists of four words. This relatively small size may cause the record tables (cf. section 5) describing records with many array elements to occupy too much buffer space and to slow down the transfer process unnecessarily much.

However, it is possible to generate special versions of the SAMBA-compiler with a greater working area. If this is wanted, the SLIP name e69 must be redefined to the new size of the working area. This must be done after the message

redefine wanted loading parameters

has been printed out. Only the following restrictions should be considered:

$$4 \leq e69 \leq 40$$

Note that the redefinition of e69 may cause the number of available track-places to be decreased accordingly (cf. ref. 2 section 11.3).